Tampereen ammattikorkeakoulu
Ammatillinen opettajakorkeakoulu

Mika Tuomainen

Kehittämishanke

Ohjelmoinnin perusteiden opetuksen ja opiskelun apuvälineet

Teoriataustaa ja arviointimalli

Työn ohjaaja Sirpa Levo-Aaltonen
Tampere 01/2011

Tampereen ammattikorkeakoulu
Ammatillinen opettajakorkeakoulu
Opettajankoulutuksen kehittämishanke

Tuomainen, Mika
Ohjelmoinnin perusteiden opetuksen ja opiskelun apuvälineet – Teoriataustaa ja arviointimalli
40 sivua + 2 liitesivua
Tammikuu 2011
Työn ohjaaja Sirpa Levo-Aaltonen

TIIVISTELMÄ

Työssä käsiteltiin ohjelmoinnin opetukseen tarkoitettuja visualisointivälineitä sekä ohjelmointiharjoitusten suorittamisen apuvälineitä. Opettaja voi hyödyntää visualisointiapuvälineitä omassa opetustyössään havainnollistamaan opetusta ja korvaamaan perinteisiä menetelmiä. Havainnollistamisen lisäksi apuvälineitä on tarjolla myös ohjelmointiharjoitusten suorittamiseen. Välineet tarjoavat tällöin oppilaille valmiin ympäristön harjoitusten suorittamiseen. Tämä on tärkeä seikka etenkin ohjelmoinnin opiskelun alussa. Apuvälineissä ympäristöt löytyvät valmiina ja opiskelussa voi keskittyä heti itse asiaan, eikä onnistuminen jää ainakaan kiinni tarvittavan ohjelmointiympäristön puuttumisesta.

Kehittämishankkeessa kuvattiin arviointimalli ohjelmoinnin opetuksen apuvälineiden arviointiin. Opettaja voi käyttää arviointimalliin määriteltyjä muuttujia apuna etsiessään ja valitessaan sopivaa välinettä halumaansa käyttötarkoitukseen. Arviointimalliin määriteltiin muuttujia, joiden avulla eri apuvälineiden arviointia on mahdollista suorittaa. Tässä työssä arvioitiin kahta apuvälinettä ja käytettiin näiden arviointiin määriteltyä arviointimallia. Arviointi kohdistui tiettyihin ennalta asetettuihin tarpeisiin arviointimallin muuttujien mukaisesti. Näitä tarpeita olivat mm. mihin väline on tarkoitettu, välineiden kattavuus, verkkokäyttöisyys tai asennettavuus omalle koneelle, maksullisuus tai ilmaisuus, välineen hallinta ja elinkaari.

On huomattava, että tässä työssä rajauduttiin vain niihin apuvälineisiin, jotka soveltuivat ohjelmoinnin perusteiden opetuksen ja opiskelun vaatimuksiin. Osa läpikäytävistä esimerkkivälineistä voi tukea pidemmällekin meneviä ominaisuuksia mutta niitä ei tarkasteltu. Hankkeen tuloksista on hyötyä kaikille ohjelmoinnin opettajille ohjelmoinnin apuvälineiden valinnassa ja arvioinnissa.

Asiasanat: Ohjelmointi, havainnollistaminen, visualisointi, algoritmianimaatio

Sisällysluettelo

1 Johdanto

Ohjelmoinnin perusteiden opetus on aluksi ohjelmoinnin perusrakenteiden ja algoritmien opiskelua. Opetus aloitetaan yleensä ensin pseudokielisellä ohjelmoinnilla, joka muistuttaa puhekieltä. Tällöin kuvataan ohjelman toimintaa siten, että eri ohjelmointikielen syntaksit eli tavat esittää ohjelmia ohjelmointikielillä piilotetaan ja jäljelle jää vain ohjelman perusrakenne. Varsin pian mukaan tulevat kuitenkin varsinaiset ohjelmointiin liittyvät rakenteet ja algoritmit. Tyypillisesti vasta tämän jälkeen valitaan ohjelmointikieli, jolla ohjelmointia opiskellaan ja opetetaan eli käytännössä aletaan suorittaa varsinaista ohjelmien ohjelmointia.

Suuri osa ohjelmoinnin perusteiden opetuksesta koostuu eri perusohjelmointirakenteiden sekä perusalgoritmien opettamisesta. Nämä ovat myös yksi ohjelmoinnin opetuksen kriittisimmistä vaiheista jatkon kannalta, sillä nämä samat perusrakenteet ja algoritmit toistuvat samankaltaisina muissakin jatkossa opiskeltavissa ohjelmointikielissä. Lähiopetuksessa perusrakenteiden ja algoritmien käyttöä havainnollistetaan yleensä käymällä niitä läpi yksikertaisten ohjelmaesimerkkien avulla. Tällöin opettaja esittelee av-tekniikan avulla näiden ohjelmien ja esimerkkien suorittamista sekä niiden etenemistä ja eri tiloja vaihe vaiheelta. Tämä lähiopetus on ohjelmointia "kädestä pitäen" opettamalla.

Tässä työssä käsiteltävät ohjelmoinnin opetuksen visualisointivälineet tarjoavat apua edellä kuvattuun havainnollistamistarpeeseen. Opettaja voi hyödyntää näitä apuvälineitä omassa opetustyössään havainnollistamaan opetusta ja korvaamaan perinteistä ”kalvo-tussi-piirtoheitin”-kombinaatiota. Visualisointivälineet toimivat myös opiskelijan itsenäisen opiskelun tukena vastaavana havainnollistamisapuvälineenä kuin lähiopetuksessa. Ohjelmointia opetetaan nykyisin paljon myös verkko-opetuksena. Tällöin lähiopetusta ei välttämättä ole lainkaan ja lähiopetuksessa käytettävä ”kädestä pitäen” opettaminen ei ole mahdollista. Erilaisilla visualisointivälineillä on mahdollista korvata ainakin osa lähiopetuksen puuttumisesta ja auttaa opiskelijaa ymmärtämään itsenäisessä opiskelussa paremmin ohjelmointirakenteiden ja -algoritmien toimintaa.

Havainnollistamisen lisäksi apuvälineitä on tarjolla myös ohjelmointiharjoitusten suorittamiseen. Välineet tarjoavat tällöin oppilaille valmiin ympäristön harjoitusten

suorittamiseen. Tämä on tärkeä seikka etenkin ohjelmoinnin opiskelun alussa. Ohjelmointiympäristöjen rakentaminen, joissa harjoituksia tehdään, voi olla joskus varsin hankalaa. Apuvälineissä ympäristöt voivat löytyä valmiina. Tämä puolestaan edesauttaa sitä, että opiskelussa voi keskittyä heti itse asiaan, eikä onnistuminen jää ainakaan kiinni siitä, ettei opiskelija ole saanut tehdyksi tarvittavaa ohjelmointiympäristöä.

Ohjelmoinnin apuvälineiden käytöstä voi siis hyötyä sekä opettaja että oppilas. Opettaja voi käyttää apuvälineitä oman opettamisen tukena ja opiskelijat vastaavasti itsenäisen opiskelun tukena.

Tämän kehittämishankkeen tavoitteena on kuvata arviointimalli sellaisten ohjelmoinnin apuvälineiden arviointiin, joita voidaan hyödyntää ohjelmoinnin perusteiden opetuksessa. Lisäksi käydään teoriataustaa aiheeseen liittyen. Opettaja voi käyttää arviointimalliin määriteltyjä muuttujia apuna etsiessään ja valitessaan sopivaa välinettä halumaansa käyttötarkoitukseen. Arviointimalliin määritellään muuttujia, joiden avulla eri apuvälineiden arviointia voidaan suorittaa. Arviointi kohdistuu välineiden soveltuvuuteen tiettyihin ennalta asetettuihin tarpeisiin. Näitä tarpeita ovat mm. mihin väline on tarkoitettu, välineiden kattavuus (mitä perusrakenteita/mitä väline algoritmeja kattaa), soveltuvuus itsenäiseen opiskeluun, verkkokäyttöisyys/asennettavuus omalle koneelle, maksullisuus/ilmaisuus jne.

On huomattava, että tässä työssä rajaudutaan vain niihin apuvälineisiin, jotka soveltuvat ohjelmoinnin perusteiden opetuksen ja opiskelun vaatimuksiin. Osa läpikäytävistä esimerkkivälineistä voi tukea pitemmällekin meneviä ominaisuuksia, mutta niitä ei tarkastella.

Hankkeen tuloksista on hyötyä kaikille ohjelmoinnin opettajille ohjelmoinnin apuvälineiden valinnassa ja arvioinnissa.

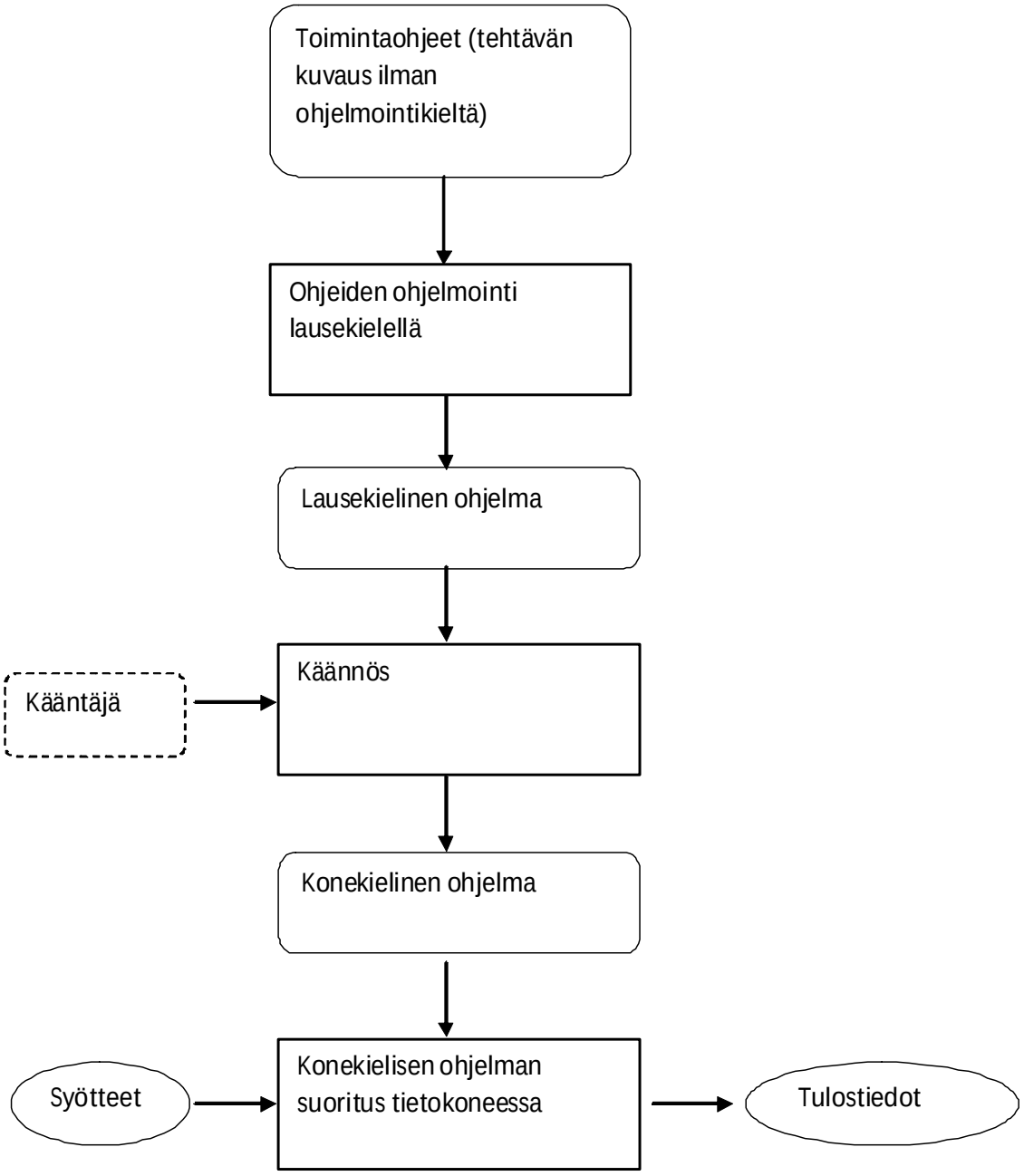
2 Ohjelmointi

2.1 Ohjelmoinnin taustoja

Lyhyesti ilmaistuna ohjelmointi tarkoittaa tietokoneelle tai vastaavalla laitteelle annettavia toimintaohjeita (käskyjä) jonkin tehtävän ratkaisemiseksi. Yksinkertaisin ohjelmointikieli on konekieli, jonka toimintaohjeet tietokoneen prosessori osaa tulkita suoraan. Aluksi ohjelmointi olikin suoraan konekielellä ohjelmointia. Se oli kuitenkin hidasta ja erittäin virhealtista, sillä ohjelmien monimutkaisuuden kasvaessa tarvittiin hyvin paljon konekielisiä käskyjä. (Boberg ym. 2000.)

Ohjelmoinnin helpottamiseksi kehitettiin korkeamman tason kieliä eli lausekieliä. Ne ovat koneista riippumattomia ohjelmointikieliä. Lausekielet eivät ole tiukasti sidottuja koneen rakenteeseen ja toimintaan, vaan ne on pyritty suunnittelemaan ja toteuttamaan niin, että niiden käyttö on ihmiselle mahdollisimman helppoa ja luontevaa. Jokaisella lausekielisellä lauseella voidaan esittää paljon suurempi osa käskyjä kuin konekielisillä käskyillä (itse asiassa yksi lausekielinen käsky voi sisältää useita konekielelle käännettyjä tai tulkittuja käskyjä).

Tietokone, laite tai prosessori ei kuitenkaan ymmärrä lausekielisiä käskyjä suoraan, vaan lausekieliset ohjeet on muunnettava konekielisiksi. Tämä muuntaminen tapahtuu joko kääntämällä koko lausekielinen ohjelma kerralla konekieliseksi tai tulkitsemalla lausekieliset ohjeet lause kerrallaan sitä mukaa kun ohjelmaa suoritetaan. Tästä prosessista on kuvaus alla kuviossa 1.



Kuvio 1. Toimintaohjeiden kääntäminen konekielelle (mukailtu Boberg ym. 2000).

Ensin ratkaistavalle tehtävälle määritellään tehtävänkuvaus eli toimintaohjeet ilman ohjelmointikieltä. Tämän jälkeen ohjelmoija ohjelmoi toimintaohje lausekielellä. Tämän ohjelmointityön tuloksena syntyy lausekielinen ohjelma, jota kone ei siis vielä suoraan ymmärrä. Tämän vuoksi lausekielinen ohjelma pitää kääntää kääntäjän avulla. Käännöstyön tuloksen syntyy konekielinen ohjelma, jota tietokone ymmärtää ja jonka

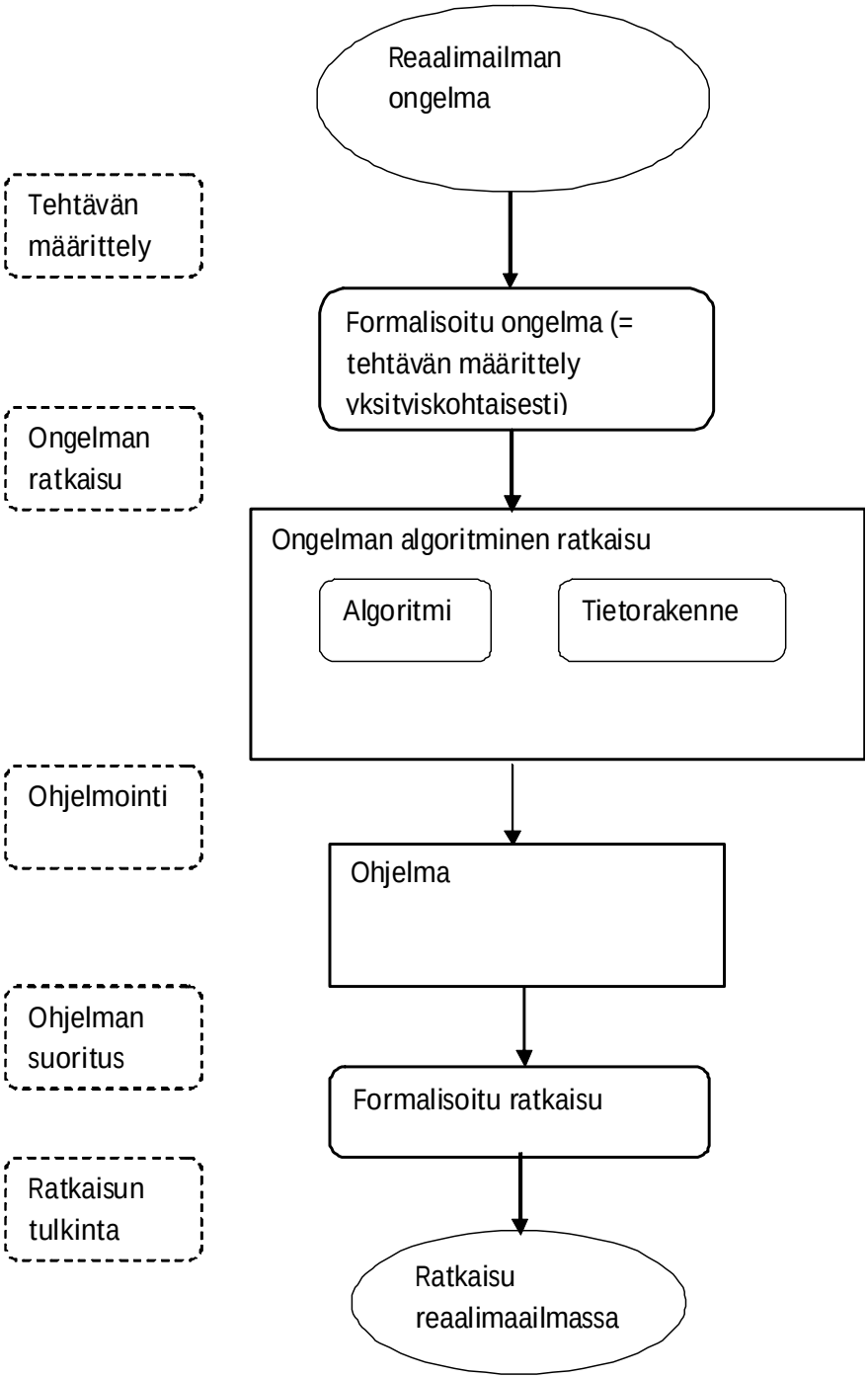
se näin voi suorittaa. Kuvassa näkyy suorituksessa myös syötetietojen antaminen ohjelmalle ja ohjelman suorituksen tuloksena näytettävät tulostiedot. Tulkitseminen tapahtuu siten, että kääntäjän paikalla toimii tulkitsija. Erona on, että kääntäjä kääntää koko ohjelman kerralla mutta tulkitsija lause kerrallaan sitä mukaan kun ohjelma etenee (aina ei ole välttämätöntä kääntää koko ohjelmaa kerralla, vaan tarpeen mukaan).

Mitä nämä ohjelmointikielien toimintaohjeet tai käskyt sitten ovat? Yksinkertaisesti tähän voidaan vasta, että ne ovat algoritmeja. Algoritmi onkin yksi ohjelmoinnin keskeisimmistä käsitteistä. Algoritmia voidaan kuvata seuraavasti (Boberg ym. 2000):

- Algoritmi on jonkin tehtävän suorittamiseksi tarvittavien toimenpiteiden joukon kuvaus.
- Algoritmi on menetelmä, jonka mukaisesti tehtävä suoritetaan.
- Algoritmi muodostuu toimenpiteistä, joiden onnistunut suoritus johtaa annetun tehtävän tekemiseen.

Algoritmin sisältämä toimenpiteiden joukko on yleensä järjestetty niin, että toimenpiteet suoritetaan yksi kerrallaan peräkkäin. Algoritmin suorittamiseen tarvitaan subjekti, joka voi olla esimerkiksi ihminen, robotti tai tietokone. Jotta subjekti osaa toimia oikealla tavalla toimintaohjeiden mukaisesti, on toimintaohjeet annettava subjektin ymmärtämällä tavalla. Näin tietokoneelle esitettävät ohjeet on annettava sen ymmärtämässä konekielisessä muodossa.

Algoritmeilla pyritään siis antamaan ohjeet jonkin tietyn asian tai ongelman ratkaisemiseksi. Alla on kuvattu (kuvio 2) reaalimaailman ongelman ja ongelman ratkaisun suhdetta ohjelmoinnissa.



Kuvio 2. Reaalimaailman ongelman ja ongelman ratkaisun suhde ohjelmoinnissa (mukailtu Boberg ym. 2000).

Alla on vielä esimerkki algoritmista teen keittämiseksi eli teen keiton ohjeet. Esimerkistä havaitaan myös se seikka, kuinka tarkalle tasolle ohjeet konetta varten

täytyy tarkentaa (vaikka esimerkki ei vielä edes mene tarpeeksi tarkalle koneen ymmärtämälle tasolle).

```
1.    keitä vettä
2.    laita pussi kuppiin
3.    kaada vettä kuppiin
```

Kohta 1 tarkennettuna:

```
1.1   Täytä vesipannu
1.2   Aseta pannu liedelle
1.3   Lämmitä kunnes vesi kiehuu
1.4   Siirrä pannu sivuun liedeltä
```

Kohta 1.1 tarkennettuna:

```
1.1.1 Aseta vesipannu hanan alle
1.1.2 Avaa vesihana
1.1.3 Odota kunnes pannu on täynnä
1.1.4 Sulje vesihana
```

Tätä tarkennusta pitää jatkaa riittävälle tasolle jokaisen kohdan osalta.

2.2 Ohjelmoinnin peruskäsitteitä

Algoritmien kuvaamiseen lausekielellä (ja myös konekielellä) eli varsinaiseen ohjelmointiin tarvitaan useita erilaisia ohjelmoinnin elementtejä. Näitä käydään läpi seuraavissa aliluvuissa.

2.2.1 Normaali teksti

Alla on esimerkki ohjelman koodista, siis ohjelmoidusta ohjelmasta. Kuten esimerkistä näkyy, käytetään ohjelmoinnissa normaaleja kirjaimia ja merkkejä eli käytännössä kuka tahansa joka osaa lukea, osaa lukea myös ohjelmakoodia. Merkkien ymmärtäminen ohjelman kontekstissa on eri asia.

```
// Ohjelma tulostaa tekstin Hello world!
class Hello
{
        public static void main(String[] args)
        {
            System.out.println("Hello world!");
        }
}
```

2.2.2 Muuttujat ja sijoitusoperaatiot

Keskeisiä ohjelmoinnin peruskäsitteitä ovat muuttuja (variable) ja sijoitusoperaatio (assignment operation). Ohjelmoinnissa muuttuja on arvon säilytyspaikka. Wikla (Wikla 2003) kuvaa muuttujaa "laatikkona, jonka kyljessä on nimi. Laatikon sisältöä voi tutkia, kysymällä muuttujan arvoa (mitä laatikossa on sisältönä) ja sisältöä voi muuttaa asettamalla sinne uutta sisältöä. Uuden sisällön asettamista laatikkoon kutsutaan sijoitusoperaatioksi. Sijoitusoperaatiolla annetaan muuttujalle arvo tai korvataan muuttujan vanha sisältö uudella. Uusi arvo on usein jonkin laskutoimituksen, ns. lausekkeen arvo. Lauseke voi sisältää muuttujien arvoja ja laskutoimituksia".

Muuttujien määrittely

Muuttujat on aina määriteltävä eli niille on annettava jokin tietty tietotyyppi. Alla on esimerkkejä muuttujien määrittelyistä.

```
int lkm;
double paino;
boolean oikein;
String etunimi;
```

Muuttuja lkm on kokonaislukutyyppinen. Sen arvoksi voidaan asettaa vain kokonaislukuja. Muuttuja paino on desimaalilukutyyppinen, jonka arvoksi voidaan antaa desimaalilukuja (liukulukuja). Muuttuja oikein on totuusarvoinen, joka saa arvokseen joko true tai false. Muuttuja etunimi on merkkijonoarvoinen.

Sijoitusoperaatiot

Edellisessä kappaleessa kuvatuille muuttujille voi sijoittaa arvoja sijoitusoperaatioilla

```
lkm = 6;
paino = 92.57;
oikein = true;
nimi = "Mika Tuomainen";
```

Seuraavaksi käydään esimerkkien avulla hieman syvällisemmin sijoitusoperaatioita. Annetaan muuttujalle nimeltä eka arvo 3

```
eka = 3;
```

Annetaan muuttujalle nimeltä eka nyt arvo 1 (arvo 3 korvautuu arvolla 1)

```
eka = 1;
```

Sijoitusoperaatio + laskutoimitus + eka-muuttujan käyttö, muuttuja toka saa arvokseen laskutoimituksen tuloksen (9)

```
toka = 9 * eka;
```

Sijoitusoperaatio + laskutoimitus + kahden muuttujan käyttö, kolm saa arvon 1000 eli (1+9) * 100

```
kolm = (eka + toka) * 100;
```

Seuraavanlainen muuttujien käyttö on myös sallittua ja yleistä ohjelmoinnissa

```
eka = eka + 1;
```

Edellä eka saa arvon 2 (1+1), ekan vanhaan arvoon lisätään yksi ja saatu summa sijoitetaan muuttujan eka uudeksi arvoksi.

Sijoitusoperaatio kannattaa lukea "saa arvokseen", ei "on", Varsinkin viimeinen esimerkki "eka saa arvokseen eka plus yksi" ilmaisee tämän näkökulman hyvin.

Muuttujien tilat

Muuttujilla on ohjelmointialgoritmeissa kullakin ajan hetkellä tietty. tila (state). Wiklan mukaan (Wikla 2003) ohjelman ”viisaudessa” kyse on siitä, että ohjelmoija on osannut

rakentaa algoritmille sellaisen tilojen ketjun, joka vaikuttaa ohjelman käyttäjästä viisaalta.

Esimerkki tilojen kuvaamisesta (mukailtu lähteestä Wikla 2003):

Alkutilassa on määritelty kolme muuttujaa eka, toka ja kolm:

```
      int eka;
      int toka;
      int kolm;

tila:     |__?____|  |__?____|  |__?____|
            eka        toka       kolm

Seuraavaksi muuttujalle eka annetaan arvo 3 ja sen tilaa päivittyy
tätä sijoitusta vastaavaksi:

      eka := 3;

tila:     |__3____|  |__?____|  |__?____|
            eka        toka       kolm

Seuraavaksi muuttujalle toka annetaan arvo 2 ja sen tilaa päivittyy
tätä sijoitusta vastaavaksi:

      toka := 2;

tila:     |__3____|  |__2___|  |__?____|
            eka        toka       kolm

Seuraavaksi muuttujalle kolm annetaan laskutoimituksen arvo 6 ja sen
tilaa päivittyy tätä sijoitusta vastaavaksi:

     kolm := eka * toka;

tila:     |__3____|  |__2___|  |__6_|
            eka        toka       kolm

Seuraavaksi muuttujalle eka annetaan arvo 111 ja sen tilaa päivittyy
tätä sijoitusta vastaavaksi:
     eka := 111;

tila:     |__111____|  |_2___|  |__6_|
            eka        toka       kolm

Edellinen esimerkki kuvaa hyvin sitä, että muuttujat ovat tiedon
säilytyspaikkoja ja niillä on aina tietty sen hetkinen tila.
```

Syöttö- ja tulostusoperaatiot

Algoritmi liitetään ympäristöönsä ns. syöttö- ja tulostusoperaatioin (Wikla 2003). Muuttujien arvot voivat olla etukäteen ohjelmakoodissa annettuja mutta tällöin kyseessä on staattinen ratkaisu. Käytännössä muuttujat saavat arvoja syöttöoperaatiossa eli tiedon lukemisen seurauksena. Syöttöoperaatio voi olla arvon saaminen ohjelman käyttäjältä

tai esimerkiksi lukeminen tiedostosta. Tulostusoperaatiossa siirretään ohjelman tuottamia tuloksia eli muuttujien tiettyjä tiloja jollekin tulostuslaitteelle.

Laskutoimitukset (operaattorit)

Laskutoimituksia suoritetaan operaattoreiden avulla. Kokonaisluvuille ja desimaaliluvuille on käytettävissä normaalit matemaattiset laskutoimitukset:

```
+ : yhteenlasku
- : vähennyslasku
* : kertolasku
/ : jakolasku
% : jäännösjako
```

Operaatio "/" tarkoittaa kokonaisjakoa, jos sekä osoittaja, että nimittäjä ovat kokonaislukutyyppiä. Aina muulloin kyseessä on liukulukujakolasku.

Operaatio "%" on kokonaislukujen jäännösjako. Tuloksen etumerkki on sama kuin osoittajan etumerkki, nimittäjän etumerkillä ei ole vaikutusta tuloksen etumerkkiin.

2.2.3 Ohjelmoinnin perusmenetelmiä

Ohjelmoinnissa on useita perusmenetelmiä. Näitä ovat mm. suoritusjärjestys, valinta ja toisto. Lisäksi käytetään erilaisia rakenteita muuttujien tietojen säilyttämiseen. Tällaisia tietorakenteita ovat esimerkiksi taulukot ja listat.

Suoritusjärjestys

Oletussuoritusjärjestys on peräkkäisyys. Peräkkäin kirjoitetut operaatiot suoritetaan peräkkäin, siinä järjestyksessä kuin ohjelmoija on ne ohjelmaan ohjelmoinut. Esim. teen keitossa mennään tietyssä järjestyksessä eteenpäin ja ohjelmoijan on ymmärrettävä kirjoittaa ohjelmansa siten, että käskyt tulevat oikeassa järjestyksessä (ei esim. siten että pannu laitetaan liedelle ennen pannun täyttämistä vedellä).

Ohjelman suoritusjärjestystä voidaan ohjailla erilaisilla rakenteilla, joista tyypillisimpiä ovat valinta- ja toistorakenteet.

Valinta

Valinnassa käydään läpi erilaisia kriteerejä sisältäviä vaihtoehtoja. Kun vaihtoehdon tietty kriteeri täyttyy, tehdään tiettyjä vaihtoehdon alle määriteltyjä toimintoja. Esim. teen keittoon liittyen:

```
jos haluan teetä
      keitän teetä
jos haluan kahvia
      keitän kahvia
muuten
      juon vettä
```

Valintaa varten voi olla myös määritelty ennakkoon tietty joukko vaihtoehtoja ja tietyt kriteerit näille vaihtoehdoille. Myös tällöin kun ehto täyttyy, valitaan kyseisen ehdon alla olevat toiminnot, esim. teen keitossa voisi valintana olla teen valinta:

```
valitse jokin vaihtoehdoista 1-3
      1:
            yellow label
      2:
            earl grey
      3:
            green tea
```

Seuraavaksi käydään yleisellä tasolla läpi, miltä edelliset valinta-rakenteet näyttävät ohjelmointikielellä ilmaistuina.

Ensimmäisessä vaihtoehdossa kriteerit tarkastetaan if-ehtolauseen avulla. Jos suluissa oleva ehto täytyy (siis on tosi) mennään if-rakenteen alla oleviin toimintoihin, ellei ehto täyty, mennään toiseen if-haaraan ja ellei tämäkään ehto täyty niin kaikissa muissa tapauksissa mennä else-haaraan:

```
if (ehto1)
{
    ehto1 edellä voisi olla esim. haluanko teetä, jos haluan eli
    ehto1 on tosi, mennään tähän alle ja aletaan keittelemään
    teetä, ellei ehto ole tosi, niin siirrytään seuraavaan if-
    haaraan

}
if (ehto2)
{
    ehto2 edellä voisi olla esim. haluanko kahvia, jos haluan
    eli ehto2 on tosi, mennään tähän alle ja aletaan
    keittelemään kahvia, ellei ehto2 ole tosi, niin siirrytään
    seuraavaan else-haaraan
}
else
{
      tähän else-haaraan mennään aina, kun ehto1 tai ehto2 eivät
      ole tosia eli en halua keittää teetä enkä kahvia vaan juon
      tällöin aina vettä
}
```

Toisessa valinta-rakenteessa edellä valittiin mitä teetä aletaan keittää. Vaihtoehdot ovat tiedossa etukäteen, jolloin voidaan käyttää switch-rakennetta. Käyttäjälle ilmoitetaan aluksi vaihtoehdot, joista valinta voidaan tehdä. Tämän jälkeen valinnan perusteella mennään switch-rakenteessa valittuun vaihtoehtoon

```
Valitse vaihtoehdoista 1-3 millaista teetä haluat keittää.
(Vaihtoehto 1: yellow label, vaihtoehto 2: earl grey, vaihtoehto
3: green tea)

switch (valinnan numero)
{
  case 1:
      keitetään yellow labelia
  case 2:
      keitetään earl greytä
  case 3:
      keitetään green teetä
}
```

Toisto

Toistossa tiettyä käskyä toistetaan niin kauan, kuin etukäteen on määritelty tai niin kauan, että päästään haluttuun lopputulokseen. Esim. teen keitossa veden saaminen pannuun:

- Etukäteen tiedossa oleva toistojen määrä

  ```
  toista kuusi kertaa
        laske vettä 1 dl
  ```

  lopputuloksena pannussa on 6 dl vettä.

- Niin kauan kuin haluttu lopputulos (määrä) saavutetaan:

  ```
  tarkasta onko pannussa 6 dl vettä
        jos ei, niin laske vettä 1 dl
        jos on, niin sulje hana
  ```

  lopputuloksena pannussa on 6 dl vettä.

Seuraavaksi käydään yleisellä tasolla läpi, miltä edelliset toisto-rakenteet näyttävät ohjelmointikielellä ilmaistuina

Etukäteen tiedossa olevien toistojen määrä voidaan ilmaista for-rakenteella. For-rakenteessa määritellään muuttuja (alla numero) silmukan läpikäyntiin ja sitten ehto (numero <= 6), jota tarkastellaan joka toisto-kierroksella. Lisäksi for-rakenteessa on laskuri (numero++), joka kasvattaa muuttujan numero arvoa joka silmukan kierroksella. Alla oleva ”laske 1 dl vettä” toistetaan kuusi kertaa:

```
for (int numero = 1; numero <= 6; numero++)
{
      laske 1 dl vettä
}
```

Sama rakenne ”niin kauan kuin tietty tulos saavutetaan” menetelmällä voidaan toteuttaa while-rakenteella. While-rakenteessa määritellään ensin muuttuja (numero), kuten for-silmukassakin. Tämän muuttuja arvoa tarkastellaan while-ehdossa (numero <=6) ja jos ehto on tosi, mennään while-silmukan sisään. Siellä muuttujaa (numero) kasvatetaan aina yhdellä. Silmukka pyörii niin kauan, kuin while-rakenteen ehto on tosi.

```
int numero = 0;
while (numero <= 6)
{
    laske 1 dl vettä
    numero++;
}
```

Tietorakenteita

Muuttujien arvojen säilyttämiseen voidaan käyttää useita erilaisia tietorakenteita kuten taulukot, listat ja puu-rakenteet. Tässä yhteydessä esitellään vain taulukkorakennetta (array), sillä se on tyypillisesti ensimmäinen rakenne, johon ohjelmoinnin perusteissa tutustutaan.

Edellä muuttujien yhteydessä muuttujia käytettiin siten, että niihin liittyi aina täsmälleen yksi arvo kerrallaan. Tämä lähestymistapa ei kuitenkaan ole aina mahdollista reaalimaailmassa, sillä yleensä joudutaan käsittelemään suuria tietojoukkoja. Esimerkiksi jos henkilöitä käsiteltäessä henkilöitä voi olla tuhansia ja jokaiselle oman muuttujan määritteleminen erikseen ei ole enää järkevää. Tällöin voidaan käyttää apuna taulukkotietorakennetta. Tällaisessa rakenteessa kuvataan useaa samantyyppisestä tiedosta muodostuvaa tietojoukkoa. Taulukon etuna on, että siihen voidaan tallentaa samantyyppisiä muuttujia samalle nimelle. Esimerkiksi jos annetaan laulukilpailussa pisteitä osallistujille, voidaan ne tallentaa taulukkoon yhteen ja samaan muuttujaan.

Taulukko koostuu peräkkäisistä tallennuspaikoista ja näiden tallennuspaikkojen arvoja kutsutaan alkioiksi. Alkioiden tallennuspaikat on numeroitu ja tätä järjestysnumeroa kutsutaan indeksiksi. Taulukossa oleviin muuttujien arvoihin (alkioihin) päästään käsiksi taulukon nimen ja taulukon indekseillä. Alla on esimerkki tällaisesta lähestymistavasta verrattuna yksittäisten muuttujien käsittelyyn.

Yksittäisessä muuttujien käsittelyssä esimerkiksi päivän lämpötiloja tallennettaessa joudutaan määrittelemään niin monta muuttujaa kuin on mittauskertoja:

```
int lampotila1, lampotila2, lampotila3..
```

Tietoja tallennettaessa joudutaan puolestaan käsittelemään jokaista muuttujaa erikseen:

```
int lampotila1 = 10;
int lampotila2 = 11;
int lampotila3 = 9;
…
```

Taulukossa sama voidaan esittää seuraavasti:

määritellään taulukko

```
int lampotila[];
```

taulukkoa voi käsitellä seuraavasti

```
int lampotila[0] = 10;
int lampotila[1] = 11;
int lampotila[2] = 9;
…
```

Käytännössä taulukko näyttä siten seuraavalta:

Indeksi:	0	1	2	…
Alkio:	10	11	9	…

Nyt edellä esimerkissä käsiteltiin vain muutamaa muuttujaa. Todellisuudessa taulukoiden käytön etu tulee esille, kun alkioita on vaikkapa tuhansia kappaleita. Taulukossa näitä tietoja voidaan käsitellä jokaista samalla algoritmilla eikä jokaisen muuttujan käsittelyyn tarvitse omaa muuttujan käsittelijää. Esimerkiksi taulukon alkioiden järjestäminen nousevaan tai laskevaan järjestykseen, kuten muutkin lajittelut ovat jo osa ohjelmoinnin perusteita.

3 Ohjelmoinnin opettaminen ja opiskelu

3.1. Taustaa

Ohjelmointi opetettavana asiana on haasteellista, sillä ohjelmointitaito koostuu useista eri osa-tekijäistä. Ohjelmointitaidot vaativat käsitteiden, strategioiden ja käytännön taitojen yhdistämistä. Opetuksessa on huomioitava, että ohjelmointiin kuuluu useita eri vaiheita ja opiskelija tarvitsee eri vaiheissa erityyppisiä tietoja ja taitoja. Monet taidot ovat tarpeen kaikissa vaiheissa, esim. ohjelmointikäsitteiden ymmärtäminen pitää hallita ongelmaa analysoidessa, ohjelmaa suunnitellessa, suunnitelman toteuttamisessa jollakin ohjelmointikielellä sekä testauksessa ja virheiden etsinnässä. Lisäksi pitää hallita yksittäisten tietojen osalta erilaisia strategioita taitojen soveltamiseen. Lisäksi ohjelmoinnin vaiheisiin liittyy hyviä käytäntöjä ja malleja (hyvä ohjelmointitapa). (Ala-Mutka 2006.)

Ohjelmoinnin perusteiden opettamisesta noudattaa usein seuraavia vaiheita:

- Ensin on teoriaa luentomaisesti.
- Sitten opettaja käy esimerkkejä läpi perinteisesti av-menetelmillä (tussi-kalvo-piirtoheitin).
- Tämän jälkeen opetettua asiaa harjoitellaan lähiopetuksessa ja opettaja on antamassa tarvittaessa apua ja palautetta. Harjoituksissa on jokin sanallinen tehtäväksianto, joka pitää toteuttaa ohjelmoimalla algoritmi tehtävän ratkaisemiseksi. Yleensä tehtävänannossa on jo esimerkki, millainen lopputuloksen pitää olla. Tällä ei pyritä niinkään rajaamaan erilaisten ratkaisujen syntyä vaan pikemminkin rajaamaan tehtävän laajuutta ja että harjoittelu kohdistuu olennaisiin asioihin.

Edellä kuvatuissa vaiheissa on seuraavia yleisiä ongelmia:

- Opetuksessa ei ole aikaa käydä jokaisen opiskelijan kanssa erikseen läpi tehtyjä ratkaisuja.
- Jos opiskelija ei ole paikalla tunneilla, niin on riskinä, että jää jälkeen. Ohjelmointitaidon karttuminen eteen vaiheittain ja jos jotain vaiheita jää pois, on mahdotonta oppia myöhemminkään kunnolla.

- Kaikki opiskelijat eivät opi ohjelmointia ja sen logiikkaa heti, vaan se vaatii työtä ja toistoa sekä oppimista valitsemaan sopivimmat ratkaisut (em. strategia) eri algoritmeihin. Varsinkin viimeinen ongelmallista itsenäisen opiskelun osuudessa ja usein ohjelman algoritmin työstäminen voi jumiutua johonkin ongelmaa, joka estää lopullisen ratkaisun toteuttamisen. Tällöin palautteen saaminen ongelman ratkaisemiseksi on tärkeää saada nopeasti.

Havainnollistamisvälineiden avulla opettaja voi korvata av-menetelmät erilaisilla apuvälineillä. Samat apuvälineet voivat toimia myös opiskelijan itsenäisen opiskelun tukena, esimerkiksi jos opiskelija on ollut poissa tunnilta tai hän tarvitsee enemmän aikaa esimerkkien läpikäymiseen kuin mitä tunnilla on mahdollista. Harjoitusten tekemisessä apuvälineiden avulla voidaan keskittyä itse opetettavaan osa-alueeseen. Ne myös mahdollistavat erilaisten harjoituksissa tehtävien ohjelmien ongelmakohtien paikallistamisen. Näin oppilas ei jää jumiin tiettyihin hankaliin kohtiin ja hän voi ohittaa ne selvittämällä ongelmaa itsenäisesti.

3.2 Apuvälineet

Ohjelmoinnin peruskäsitteiden omaksuminen on ohjelmointitaitojen perusedellytys. Tutkimusten mukaan opiskelijoilla on kuitenkin huomattavia vaikeuksia näiden omaksumisessa (McCracken ym. 2001; Ala-Mutka 2006). Ilman peruskäsitteiden ymmärtämistä puuttuu ohjelmointitaidosta käytännössä pohja kokonaan. Tätä ei helpota tulevaisuudessa henkilökohtaisen ohjauksen väheneminen niukkojen opetusresurssien ja suurten ryhmäkokojen tai opetuksen verkkoon siirtymisen vuoksi.

Ongelmaa voidaan osin ratkaista erilaisilla ohjelmoinnin opetuksen apuvälineillä. Varsinkin apuvälineet, joita opiskelijat voivat käyttää itsenäisen oppimisen tukena, ovat tarpeellisia

3.2.1 Apuvälineet visuaaliseen havainnollistamiseen

Ohjelmoijan mielessä on tietynlainen käsitys ohjelman toiminnasta eli ratkaisusta, jota hän lähtee toteuttamaan tai jota hän suunnittelee. Ohjelman toimintaa voidaan

havainnollistaa visuaalisesti. Havainnollistamisen avulla ohjelmoija voi ymmärtää paremmin, miksi ohjelma toimii, kuten toimii. Sen avulla voidaan myös helpottaa syiden etsimistä, miksi ohjelma ei toimi niin kuin ohjelmoija on ajatellut.

Ari Korhonen esittää syitä havainnollistamisen tarpeeseen (Korhonen 2005):

> "Motivaationa sekä varhaisissa ohjelmistojen havainnollistamiseen pyrkivissä esityksissä että nykyaikaisissa välineissä on siis ohjelman toiminnan ymmärtäminen ja selittäminen. Toisin sanoen tavoitteena on, että ihminen kykenee seuraamaan sitä logiikkaa, jolla ohjelma suoriutuu sille asetetusta tehtävästä (tai vaihtoehtoisesti miksi se ei suoriudu). Tästä seuraa, että ohjelmien havainnollistamiseen kykeneviä työkaluja voidaan soveltaa mm. ohjelmien kehittämiseen, niiden suunnitteluun ja määrittelyyn, testaukseen, virheenjäljitykseen sekä opiskeluun ja opetukseen."

Tietokoneohjelmien visualisoinnilla tarkoitetaan ohjelman tai algoritmin esittämistä graafisesti. Visualisoinnin tarkoituksen on auttaa käyttäjää ymmärtämään mitä ohjelma tekee, miksi se tekee niin, miten se toimii ja mitä suorituksesta seuraa (Wiggins 1998). Visualisointivälineiden avulla voidaan konkretisoida ohjelmien ja algoritmien suoritusta. Tämä konkretisointi puolestaan helpottaa ohjelmien ymmärtämistä ja tätä kautta oppimistuloksia. Ohjelmasuorituksen visualisointia voidaan käyttää näin myös menetelmänä ohjelmoinnin perusteiden opettamiseksi aloittelijoille (Kaila ym. 2009).

Visualisointi voidaan jakaa kahteen osaan: staattiset ja dynaamiset visualisaatiot. Staattisia visualisaatioita ovat esimerkiksi vuokaaviot, ohjelmointiin suunnattujen tekstinkäsittelyohjelmien visuaaliset koodaukset (sisennykset, varatut sanat eri väreillä jne.) ja ohjelman luokkien välisten riippuvuuksien esittäminen ohjelmointiympäristöissä tai graafisesti. Näillä pystytään kuitenkin auttamaan vain ohjelman rakenteen ymmärtämistä, ne eivät visualisoi varsinaista ohjelman toimintaa. (Korhonen 2005.)

Dynaamisissa visualisaatioissa voidaan tarkastella ohjelman etenemistä vaiheittain ja mitä eri tiloja ohjelmalla on näissä eri vaiheissa. Dynaaminen visualisointi pitää sisällään algoritmien läpikäynnin vaihe vaiheelta, muuttujien arvojen tarkastelun, muuttujien arvojen muutokset algoritmin eri vaiheissa sekä erilaisten ehto- ja toistorakenteiden läpikäynnin ja niiden vaikutukset muuttujiin ja niiden arvoihin.

Lisäksi visusalisoinnin kautta on mahdollista saada havainnollisempia virheilmoituksia kuin ohjelmistoympäristöjen kääntäjien antamat virheilmoitukset siitä, miksi ohjelma ei toimikaan niin kuin pitäisi tai miksi ohjelma ei käänny suoritettavaksi ohjelmaksi. Dynaamisella visualisaatiolla on näin mahdollista kuvata myös ohjelman rakenteen lisäksi myös ohjelman varsinaista toiminnallisuutta. (Korhonen 2005.)

Dynaaminen visualisointi voidaan jakaa vielä algoritmianimaatioihin ja algoritmisimulaatioihin (Korhonen 2005). Algoritmianimaatioissa kuvataan, kuinka algoritmi muokkaa suorituksen aikaisesti tietorakenteita, muuttuja ja ohjelman tiloja ja nämä muutokset havainnollistetaan käyttäjälle. Esimerkiksi kuinka listamuodossa oleva tietoaineisto järjestyy annetulla järjestämismenetelmällä (vaikka nimet aakkosjärjestykseen). Algoritmianimaatio on kuitenkin esitystapa, johon ei liity mitään käyttäjän kannalta vuorovaikutteisia elementtejä. Tässä mielessä algoritmisimulaatiot ovat parempia. Algoritmisimulaatioissa käyttäjällä on mahdollisuus muokata todellisia tietorakenteita oletetun algoritmin suorituksen mukaisesti graafisen käyttöliittymän kautta. Käyttäjä muokkaa tällöin todellisia tietorakenteita, jonka seurauksena myös vastaava visualisaatio päivittyy automaattisesti. Tämä mahdollistaa käyttäjälle esim. erilaisten muuttujien arvojen antamisen ja visualisointi havainnollistaa näiden käyttäjän tekemien muutosten vaikutuksen lopputulokseen. Tämä edesauttaa ohjelman toiminnan ymmärtämistä.

3.2.2 Apuvälineet harjoitusten suorittamiseen

Ohjelmointitaidot voidaan kehittää ainoastaan pitkäjänteisellä ja systemaattisella harjoittelulla. Ensimmäisten ohjelmien kirjoittamiseen ja harjoitusten tekemiseen liittyy tiettyjä opiskelijoiden yleisesti kohtaamia vaikeuksia. Näitä vaikeuksia ovat kokemusten perusteella seuraavat (Truong ym. 2005):

- Kuinka käytetään tekstieditoria, jolla ohjelmat kirjoitetaan.
- Kääntäjän asentaminen (kääntäjä on ohjelma, jolla oma kirjoitettu ohjelma käännetään varsinaiseksi tietokoneen ymmärtämäksi ohjelmaksi, ks. luku 2.1)
- Kuinka kääntäjää käytetään.
- Miten kääntäjän antamia virheilmoituksia ohjelmassa olevista virheistä tulkitaan.

- Kuinka etsitään ylipäätään ohjelmasta virheitä.

Edellä mainitut kolme ensimmäistä kohtaa ovat siinä mielessä kriittisiä, että jos niitä ei saada ratkaistua, on harjoitusten tekeminen käytännössä mahdotonta (oletusvaatimuksena on jo tietokoneen löytyminen opiskelijan kotoa). Kaksi viimeistä kohtaa taasen liittyvät palautteen antamiseen ja opiskelijalle näistä heränneisiin kysymyksiin.

Jos onnistumisen elämyksiä ei saavuteta heti ohjelmoinnin opiskelun alussa, seuraa siitä huonoja tuloksia jatkon kannalta ja jo ohjelmoinnin johdantovaiheessa opiskelijat voivat menettää luottamuksensa kykyihinsä. Tällöin on riskinä, että opiskelijat kokevat ohjelmoinnin vaikeana huonojen kokemuksien vuoksi ja saattavat lopettaa opiskelut. (Malmi ym. 2004; Ala-Mutka 2006.)

Tätäkään tilannetta ei auta se, että opiskelijaryhmät ovat suuria tai opetus tapahtuu verkossa ja lähiopetuksen antaminen on näin vaikeaa. Kaikki oppilaat eivät saa tällöin tarvitsemaansa yksilöllistä huomiointia. Ohjelmoinnin opiskelussa varsinkin alkuvaiheessa on tärkeää, että oppilailla on mahdollisuus harjoitella ympäristössä, jossa he saavat rakentavaa ja oikeaa palautetta. Palaute on tärkeää varsinkin siinä oppimisprosessin tilanteessa, jossa se saatavilla pyydettäessä (opiskelijan aloitteesta) ja sen avulla voidaan selventää kohtia, joita ei ole ymmärretty lainkaan tai asiat on ymmärretty väärin. Palautteen ja lisäavun antaminen on kuitenkin vaikeaa tiukkojen aikaresurssien tai lähiopetuksen puutteen puitteissa.

Edellä kuvattuja ongelmia voidaan ratkaista ainakin osittain harjoitusten suorittamiseen tarkoitetuilla apuvälineillä. Ensinnäkin välineiden tavoitteena on vähentää ohjelmien kirjoittamisen monimutkaisuutta ja auttaa opiskelijaa keskittymään tiettyyn opiskeltavaan aiheeseen ja nopeuttaa tällä tavoin oppimisprosessia. Apuvälineiden avulla ohjelma voi olla muuten valmis ja tehtävän ratkaisussa keskitytään tiettyyn opiskeltavaan aiheeseen (esim. ohjelmarunko on valmis ja vain ehtolausetta koodataan opiskelijan toimesta). Jos apuväline on web-pohjainen, on etuna lisäksi se, että opiskelijan ei tarvitse asentaa ja virittää ohjelmointiympäristöä omalle koneelle, joka oli yksi vaikeimmista tehtävistä ennen varsinaisen ohjelmoinnin aloittamista. Opettajalle

web-pohjaisten ratkaisujen etuna on juoheva luentomateriaalien, ohjeitten ja muiden www-linkitysten integrointi harjoituksiin.

Palautteen osalta apuvälineet voivat osata antaa virheistä selkokielisempiä ilmoituksia, kuin mitä kääntäjän antamat virheilmoitukset ovat. Apuvälineiden avulla voidaan myös paikallistaa keskeisimpiä virheitä ja antaa niihin korjausehdotuksia. Osassa apuvälineitä voi olla mahdollisuus myös antaa opettajan itse määrittelemiä virheilmoituksia ja korjausehdotuksia.

3.3 Miksi apuvälineitä käytetään niin vähän?

Edellä kuvattiin tarvetta apuvälineille ja miten niitä voidaan hyödyntää. Apuvälineiden käytöstä on saatu rohkaisevia tuloksia monissa kyselyissä ja tutkimuksissa (Levy ym. 2007; Levy ym. 2003). Mutta miksi apuvälineitä kuitenkin käytetään niin vähän? Seuraavaksi selvitetään, mitkä ovat mahdollisesti niiden käyttämisen huonoja puolia.

Levy ym. (Levy ym. 2007) mukaan syitä ovat seuraavat:

- Välineiden kehittämisessä on keskitytty usein vain välineeseen ja niistä puuttuu integrointi muihin opettajan käyttämiin välineisiin. Näin väline tuntuu irralliselta muuhun opetukseen.
- Välineet pitää huomioida pedagogisena opetusvälineenä, mutta välineiden käyttöä ei ole huomioitu opetussuunnitelmissa.
- Välineiden käyttö vaati niiden käytön opettelua ja siihen ei ole yleensä aikaa tai sitä käytetään usein perusteluna sille, ettei välineitä käytetä. Osasyynä on myös em. syiden aiheuttama muutosvastarinta uuden opiskeluun.
- Opettajilla on tunne, että välineiden vuoksi opetus muuttuu opettajakeskeisyydestä välinekeskeisesi. Opettaja ei ole näin enää luokan keskipisteenä. Tämä on varsinkin kokeneiden ja pitkään opettaneiden ongelma. Pelkona on, että välineet vähentävät opettajan roolin merkitystä.

Kuten edellä olevista seikoista huomaa, on syiden takana enemmän muut kuin tekniset seikat lukuunottamatta ensimmäistä kohtaa. Opettajan rooli ei ole enää välttämättä opettamista ryhmille luokkahuoneissa, vaan opettaja siirtyy enemmän taustalle: tuottaa

(verkko)materiaaleja, seuraa opiskelijoiden toimintaa ja reagoi tilanteisiin, mikäli jokin asia on menossa väärään suuntaan.

On kuitenkin huomattava, että välineet eivät korvaa kaikkea opettajan työtä, mutta niitä kannattaa hyödyntää kohteissa, joissa niiden käyttö on järkevää. Lisäksi opiskelijoille tuleva hyöty itsenäisessä opiskelussa välineiden avulla on huomioitava. Opettajat ovat kuitenkin avainasemassa näiden välineiden käyttöönotossa. Siksi tässäkin työssä pyritään selventämään välineitä ja mihin niitä voidaan käyttää. Lisäksi tarjotaan apua arviointiin arviointimallin kautta.

4 Arviointikehikko

Tässä luvussa kuvataan arviointimalli ohjelmoinnin apuvälineiden arviointiin. Arviointimalli kohdistuu sellaisiin apuvälineisiin, joita voidaan hyödyntää ohjelmoinnin perusteiden opetuksessa. Opettaja voi käyttää arviointimalliin määriteltyjä muuttujia apuna etsiessään ja valitessaan sopivaa apuvälinettä halumaansa käyttötarkoitukseen. Arviointimalliin on määritelty muuttujia, joiden avulla eri apuvälineiden arviointia voidaan suorittaa. Arviointi kohdistuu välineiden soveltuvuuteen tiettyihin ennalta asetettuihin kriteereihin. Näitä kriteerejä ovat Truong ym. mukaillen (Truong ym. 2005):

- Yleiset
    - käyttötapa
    - käyttöoikeudet tai käytön mahdolliset rajoitukset (lisensointi)
    - välineen hinnoittelu
    - ohjelman hallinta
    - käytön tuki
    - eri ohjelmointikielien tuki
    - ohjelman elinkaari.
- Havainnollistamisen apuvälineet
    - staattinen vai dynaaminen visualisointi
    - staattisen visualisoinnin eri tavat
    - dynaamisen visualisoinnin eri tavat
    - välineen kattavuus.
- Harjoitusten suorittamisen apuvälineet
    - mitä apua suorittamiseen löytyy valmiina
    - konfiguroitavat palautteet (yleiset palautteet, virheet)
    - opettajan vaikuttamismahdollisuus suoritettaviin harjoituksiin
    - välineen kattavuus.

On huomattava, että tässä työssä rajaudutaan vain ohjelmoinnin perusteiden opetuksen ja opiskelun vaatimuksiin. Osa läpikäytävistä esimerkkivälineistä voi tukea pitemmällekin meneviä ominaisuuksia, mutta ne eivät ole tarkastelun kohteena ja jätetään näin huomioimatta.

Seuraavana on kuvattu arviointikehikon kriteerit tarkemmin. Arviointikehikon kriteerejä on mukailtu Truong ym. (Truong ym. 2005) artikkelista Learning to Program Through the Web.

4.1Yleiset kriteerit

Yleiset kriteerit koskevat sekä havainnollistamisen että suorittamisen helpottamisen apuvälineitä mukaillen Truongin ym. (Truong ym. 2005) kriteerejä:

Välineen käyttötapa

Välineen käyttötavalla tarkoitetaan, onko väline työasemalle asennettava ohjelma vai web-pohjainen. Jos väline on työasemaohjelma, pitää se asentaa opiskelijan omalla tai koulun koneelle. Toinen vaihtoehto on, että ohjelma on web-pohjainen ja sitä käytetään www-selaimella. Työasemaohjelmien osalta on tärkeää tietää, mitä käyttöjärjestelmiä ohjelma tukee eli mille käyttöjärjestelmille ohjelma on asennettavissa. Web-pohjaisissa välineissä taasen on tärkeää selvittää, millä eri selaimilla apuvälinettä voidaan käyttää.

Käyttöoikeudet ja hinnoittelu (lisensointi)

Käyttöoikeuksien osalta on selvitettävä, onko apuvälineen käytölle olemassa rajoituksia ja millainen lisensointi apuvälineellä on. Työpöytäsovellusten osalta on hyvä selvittää, voidaanko välinettä käyttää koulussa ja kotona vai vain koulussa (eli onko ohjelma käytettävissä ainoastaan koulun koneilta vai saako ohjelman asentaa omalle koneelle). Web-pohjaisten sovellusten on selvitettävä, saako myös kotikoneelta ottaa yhteyden välinettä pyörittävään palvelimeen vai onko käyttäjän oltava koulun verkossa.

Välineen hinnoittelusta on yksinkertaisesti selvitettävä onko se maksullinen vai ilmainen. Tämä on lisäksi selvitettävä lisenssin laajuus eli voiko koulun lisenssillä asentaa ohjelman myös opiskelijan kotikoneella vai pitää siitä tehdä erillinen sopimus ja tuleeko opiskelijalla asennuksesta maksua.

Ohjelman hallinta, käytön tuki ja elinkaari

Ohjelman hallinnan osalta kannattaa selvittää, onko väline jokaisen itse hallittavissa oleva työpöytäsovellus tai itse hallittavissa oleva web-pohjainen ohjelma, joka voidaan käyttää koulun omalta palvelimelta. Web-pohjaiset ratkaisut voivat olla myös sellaisia, että ne ovat käytettävissä ainoastaan ulkopuoliselta palvelimelta tai palveluntarjoajan palvelimelta ja eivät näin ole itse hallittavissa.

Käytön tuesta kannattaa selvittää löytyykö ohjelmalle käyttäjätukea ja jos löytyy, niin onko tuki ilmaista vai maksullista. Huom. tällä haetaan ohjelman käyttöön liittyvää tukea, ei tukea ongelmatilanteisiin, joissa opiskelija jää jumiin jossain tietyssä ohjelmointiin liittyvässä ongelmassa.

Ohjelman elinkaari kertoo, kuinka kauan ohjelma on ollut saatavilla tai markkinoilla. Mitä pitempään ohjelma on ollut saatavilla, sen varmempaa on myös sen pysyminen saatavilla jatkossakin. Tämä on erityisen tärkeää web-pohjaisissa ostettavissa palveluissa.

Eri ohjelmointikielien tuki

Käytön tarpeen suhteen on selvitettävä ensin, mille ohjelmointikielelle apuvälinettä ollaan hakemassa. Tämän jälkeen pitää selvittää, tukeeko apuväline juuri tarvittavaa tai tarvittavia ohjelmointikieliä.

4.2 Kriteerit havainnollistamisen apuvälineille

Tässä luvussa käydään läpi kriteerit havainnollistamisen apuvälineille mukaillen Truongin ym. (Truong ym. 2005) kriteerejä:

Dynaaminen ja staattinen visualisaatio

Havainnollistamisen apuvälineestä on hyvä selvittää, tukeeko väline staattista vai dynaamista visualisointia vai tarjoaako väline tuen molemmille tavoille. Dynaamisen visualisoinnin osalta on hyvä tarkastella tukeeko väline:

- algoritmianimaatiota eli ohjelman etenemisen ja tilojen visualisointia
- algoritmisimulaatiota eli käyttäjän mahdollisuutta vaikuttaa simuloitavaan visualisointiin lennossa.

Konfiguroitavat palautteet

Konfiguroitavat palautteet mahdollistavat omien palautteiden antamisen ja määrittelyn. Tämä on erityisen tärkeää, kun oppilaat testaavat omia ohjelmiaan itsenäisesti. Järkevät palautteet antavat enemmän infoa kuin ohjelmointiympäristöjen kääntäjät.

Välineen kattavuus

Välineen kattavuuden arvioinnilla selvitetään, millaisia algoritmien ja ohjelmoinnin perusrakenteiden läpikäymisen mahdollisuuksia väline sisältää valmiina.

4.3 Kriteerit harjoitusten suorittamisen apuvälineille

Tässä luvussa käydään läpi kriteerit harjoitusten suorittamisen apuvälineiden arvioimiselle pohjautuen Truongin ym. (Truong ym. 2005) ajatuksiin:

Valmiit ominaisuudet

Harjoitusten suorittamisen apuvälineiden keskeiset hyödyt tulevat niiden tarjoamista valmiista osista, joita harjoitusten tekijä voi hyödyntää. Näistä tärkein on ohjelmointiympäristön tarjoaminen. Lisäksi apuväline voi tarjota valmiita harjoituksia läpikäytäväksi.

Ohjelmien kirjoittamisen monimutkaisuuden alentamisen apuvälineet

Apuväline voi myös auttaa keskittymään juuri tiettyyn opiskeltavaan aiheeseen tarjoamalla aiemmin opitut tutut osat ja ohjelmien rungot jo valmiina sekä pilkkomalla harjoitusten tekemisen osa-alueisiin (täydennä puuttuva kohta/tietty kohta -tyyliset harjoitukset).

Konfiguroitavat palautteet

Harjoitusten suorittamisen apuvälineissä on tärkeää, että virheilmoitukset ovat konfiguroitavissa ja että virheilmoitukset perustuvat myös muihin kuin kääntäjän antamiin virheilmoituksiin. Pitemmälle vietynä palaute voi olla palautetta ohjelmien laadusta ja niiden ratkaisujen oikeellisuudesta.

Opettajan vaikutusmahdollisuudet

Opettajalla voi olla välineen käyttöön vaikutusmahdollisuutena omien harjoitusten lisäämismahdollisuus, mahdollisuus antaa välineen kautta palautetta harjoituksiin sekä harjoitusten muokattavuus opiskelijoiden tason mukaan. Toisaalta opettajalla ei saata olla mitään em. mahdollisuuksia, vaan hänen on tyydyttävä apuvälineen tarjoamiin mahdollisuuksiin.

Välineen kattavuus

Välineen kattavuuden arvioinnilla selvitetään millaisia algoritmien ja ohjelmoinnin perusrakenteiden harjoituksia väline sisältää valmiina.

5 Esimerkkejä

5.1 Jeliot 3

Jeliot 3 on apuväline Java-kielisten ohjelmien visualisointiin. Se käyttää dynaamista algoritmianimaatiota ja antaa oppilaalle mahdollisuuden käydä ohjelmaa läpi askel askeleelta. Samalla se visualisoi metodikutsut, muuttujien arvot ja muutokset sekä operaatiot erillisessä omassa ikkunassa. Ohjelmat, joita apuvälineellä visualisoidaan, voidaan luoda ohjelmassa olevalla editorilla, kopioida toisesta editorista tai sitten avata valmiit ohjelmat editoriin. Oppilas voi näin kokeilla luentomateriaalien lisäksi omien ohjelmien visualisointia ja käyttää ohjemaa myös valmiina ohjelmointiympäristönä. Koodia voidaan myös muuttaa näin editorissa eli kyseessä on myös jonkin asteinen simulaatio. Tätä ei voi kuitenkaan tehdä lennosta, vaan ohjelma pitää kääntää ennen uutta animaatiota. Jeliot 3 ymmärtää suurimman osan Java rakenteista ja pystyy animoimaan ne. Kehityksen alla on myös parempi tuki olio-ohjelmointiin ja esimerkiksi periytymisen animointiin. (Jeliot 3 2008.) Olio-ohjelmointi ja periytyminen eivät kuitenkaan kuulu tämän työn tarkastelun kohteisiin.

Seuraavaksi taulukossa 1 käydään läpi Jeliot 3 apuväline luvussa 4 kuvattujen arviointikriteerien mukaisesti.

Taulukko 1. Arviointikehikkoon pohjautuva Jeliot 3 apuvälinen läpikäynti (mukaillen Jeliot 3 2008).

Jeliot 3	
Yleiset	
Käyttötapa	Jeliot3 on työasemaohjelma, joka pitää asentaa koulun koneelle tai sitten opiskelijan omaan koneeseen. Jeliot3 toimii Windows, Linux ja Mac käyttöjärjestelmissä. Vaatimuksen ohjelman ajamiseen on lisäksi Java ajoympäristö (Java Runtime Environment, JRE), joka on saatavilla ilmaiseksi. Käytännössä ohjelma toimii myös

	muissakin mahdollisissa käyttöjärjestelmissä, jos niihin on mahdollista asentaa Java ajoympäristö (JRE).
Käyttöoikeudet tai käytön mahdolliset rajoitukset (lisensointi)	Jeliot3 on ilmainen ohjelma, jota voi käyttää kotona ja koulussa maksutta. Ohjelma on ladattavissa osoitteesta http://cs.joensuu.fi/jeliot. Ohjelma on julkaistu General Public License (GPL) -lisenssin alla. Käytännössä tämä tarkoittaa, että ohjelmaa voidaan myös kehittää yhteisöllisesti niin ohjelman kehittäjien kuin opettajien ja oppilaidenkin toimesta.
Välineen hinnoittelu	Ilmainen sekä koululle että opiskelijalle.
Ohjelman hallinta	Jokaisen itse hallittavissa oleva työpöytäsovellus.
Käytön tuki	Jeliot 3 www-sivustolta (http://cs.joensuu.fi/jeliot) löytyy käyttäjäfoorumi. Lisäksi yhteyttä voi ottaa sähköpostiosoitteeseen jeliot@cs.joensuu.fi
Eri ohjelmointikielien tuki	Tarkoitettu ainoastaan Java-kielelle.
Ohjelman elinkaari	Jeliot3 on kehitetty vuodesta 2003.
Havainnollistamisen apuvälineet	
Havainnollistamistavat	Havainnollistaminen tapahtuu dynaamisen visualisointina.
Staattisen havainnollistamisen tavat	-
Dynaamisen havainnollistamisen tavat	Ohjelma käyttää pääasiassa dynaamista algoritmianimaatiota antaen oppilaalle mahdollisuuden käydä ohjelmaa läpi askel askeleelta. Ohjelma visualisoi metodikutsut, muuttujien arvot ja muutokset sekä operaatiot erillisissä omissa ikkunoissaan (ks. liite 1). Ohjelmat, joita apuvälineellä visualisoidaan, voidaan luoda ohjelmassa olevalla editorilla (ks. liite 1), kopioida toisesta editorista tai sitten avata valmiit ohjelmat editoriin. Koodia voidaan myös muuttaa

	ohjelman omassa editorissa eli kyseessä on myös jonkin asteinen simulaatio. Tätä ei voi kuitenkaan tehdä lennosta, vaan ohjelma pitää kääntää aina ennen uutta animaatiota.
Konfiguroitavat palautteet	Ohjelma ei tarjoa konfiguroitavia palautteita, eikä niitä ole mahdollista määritellä itse. Ohjelman antamat virheilmoitukset ovat Java-kääntäjän virheilmoituksia (ks. liite2).
Välineen kattavuus	Jeliot3 ymmärtää suurimman osan Java rakenteista ja pystyy animoimaan ne. Kehityksen alla on myös parempi tuki olio-ohjelmointiin ja esimerkiksi periytymisen animointiin. Jeliot3:sta voidaan käyttää näin myös ohjelmoinnin perusteiden opetuksesta eri perusohjelmointirakenteiden sekä perusalgoritmien läpikäyntiin.
Muuta	Oppilas voi kokeilla luentomateriaalien lisäksi omien ohjelmien visualisointia. Ohjelmaa voi käyttää myös valmiina ohjelmointiympäristönä eli harjoitusten suorittamisen apuvälineenä.

Jeliot 3 apuväline sopii hyvin Java-kielellä käytäviin peruskursseihin, joissa käydään läpi perusohjelmointirakenteita ja perusalgoritmeja. Opettaja voi korvata ohjelmalla perinteisen kalvoilla ohjelmien etenemisen läpikäynnin. Oppilas puolestaan voi käyttää ohjelmaa omatoimiseen opiskeluun ja myös omien ohjelmien testaamiseen. Alkuun tarvitaan jonkin verran ohjelman käyttökoulutusta mutta ohjelmaa on todella helppo käyttää.

5.2 Javala

Javala on harjoitusten suorittamisen apuväline. Javalan perusajatus on tarjota Java-ohjelmoinnissa tarvittava ohjelmointiympäristö valmiina ja oppilas voi keskittyä varsinaiseen ohjelmointiin. Javala on kehitetty Tampereen teknillisessä yliopistossa

(TTY). Javala toimii avoimena oppimisympäristönä Java-ohjelmoinnin opetuksessa. Ohjelma on web-pohjainen, jota käytetään selaimella. Sen palvelin toimii TTY:n hallinnoimana, mutta oppimisympäristö on avoin kaikille halukkaille. (Javala 2010.)

Apuvälineen avulla opiskelija voi käydä apuvälineen tarjoamia harjoituksia läpi itse haluamallaan tavalla. Web-sivusto on rakennettu siten, että siinä on aina ensin teoriaosuuksia ja näiden jälkeen teoriaa vastaavia harjoituksia.

Seuraavaksi taulukossa 2 käydään läpi Javala apuväline luvussa 4 kuvattujen arviointikriteerien mukaisesti.

Taulukko 2. Arviointikehikkoon pohjautuva Jeliot 3 apuvälinen läpikäynti (mukaillen Javala 2010).

Javala	
Yleiset	
Käyttötapa	Web-pohjainen ohjelma, jota voi käyttää selaimella. Testattu tämän dokumentin tekijän toimesta seuraavilla selaimilla: Internet Explorer, Firefox, Google Chrome.
Käyttöoikeudet tai käytön mahdolliset rajoitukset (lisensointi)	Ohjelmaa voi käyttää vapaasti ilman rajoituksia. Ohjelmaa ei kuitenkaan voi asentaa esim. koulun omalle palvelimelle, vaan sitä on käytettävä TTY:n palvelimelta.
Välineen hinnoittelu	Ilmainen opiskelijalle. Myös koulun käyttöön ilmainen mutta ensisijaisesti tarkoitettu opiskelijan itsenäiseen opiskeluun.
Ohjelman hallinta	Ohjelman hallintaan ei voi vaikuttaa vaan sitä voi käyttää ainoastaan www-palveluna. Tällaisessa toisen hallinnoimassa palvelussa on aina riskinä se, että on riippuvainen toisen osapuolen ratkaisuista (esim. palvelu voi lakata koska tahansa ja minkäänlaisia sopimuksia

	toisen osapuolen kanssa ei voi tehdä).
Käytön tuki	Ei erillistä tukea.
Eri ohjelmointikielien tuki	Ainoastaan Java-kielen opiskeluun.
Ohjelman elinkaari	Palvelu on ollut pystyssä vuodesta 2004 lähtien.
Harjoitusten suorittamisen apuvälineet	
Valmiit ominaisuudet	Ohjelma tarjoaa Java-ohjelmoinnissa tarvittavan ohjelmointiympäristön valmiina ja oppija voi keskittyä ohjelmointiin. Oppiminen tapahtuu ensisijaisesti teoriaosioiden lomassa olevia harjoituksia ja esimerkkejä tekemällä ja kokeilemalla.
Ohjelmien kirjoittamisen monimutkaisuuden alentaminen	Ohjelmointiympäristö on käytettävissä selaimella, joten oppija voi Java-kehitysohjelmistojen imuroimisen, asentamisen yms. sijasta aloittaa heti ohjelmoimaan. Harjoitukset on toteutettu siten, että lähdekoodi kootaan • valmiiksi annetuista osista, joita ei voi muokata • sekä oppilaan syöttämästä koodinpätkästä. Tämän jälkeen koodi käännetään Java-kääntäjällä, ja ajetaan erillisellä Java-virtuaalikoneella yksinkertaisella aja ohjelma-komennolla.
Konfiguroitavat palautteet	Mahdolliset käännösvirheet kerrotaan käyttäjälle Java-kääntäjän virheilmoituksina. Opiskelijan täytyy siis osata tulkita näitä virheilmoituksia.

	Jos ohjelma kääntyy oikein, niin ohjelman tulosta verrataan oikeaan vastaukseen, ja oppilaalle annetaan tämän mukaisesti palautetta ja jos virhe on tällä tasolla annetaan oppilaalle konfiguroituja palautteita.
Opettajan vaikuttamismahdollisuudet	Ei vaikutusmahdollisuutta, kaikki teoriaosuudet ja harjoitukset tulevat palvelunantajalta.
Kattavuus	Javala kattaa Java-ohjelmointikielestä enemmän kuin ohjelmoinnin perusopetuksessa on tarpeen. Kielen perusteiden lisäksi Javalassa on materiaalia seuraavista: merkkijonot, oliot, luokat, periytyminen, poikkeukset, pakkaukset, collections, I/O, grafiikka, virtuaalikone, säikeet ja synkronointi, regular expressions, appletit, mobiili Java (J2ME). Kielen perusteista käsitellään seuraavat: tietotyypit, operaattorit, kontrollirakenteet, toistorakenteet ja taulukot eli ohjelmoinnin perusopetuksen keskeiset aiheet.

Javalan perusoletuksena on, että oppilas osaa jo valmiiksi ohjelmoinnin perustaidot. Toisaalta Javalan kerrotaan kuitenkin olevan ennen kaikkea alkeisopetuksen väline, jolla on tavoitteena luoda oppijalle selkeä käsitys Java-ohjelmoinnin perusteista. (Javala 2010.)

Javalan ensimmäiset osat, joissa käsitellään Java-kielen perusteita sopivat peruskurssien apuvälineeksi hyvin. Opettaja voi antaa oppilaille näiden harjoitusten tekemisen lisätehtäväksi ja itsenäisen opiskelun tueksi. Javalassa opettajalla ei tosin ole vaikutusmahdollisuutta harjoituksiin tai muutenkaan palvelun sisältöön. Se on hyväksyttävä sellaisena kuin sitä tarjotaan.

6 Yhteenveto

Tässä työssä käytiin läpi erilaisia apuvälineitä ohjelmoinnin perusteiden opetukseen. Näistä käsiteltiin ohjelmoinnin opetukseen tarkoitettuja ohjelmien visualisointivälineitä sekä ohjelmointiharjoitusten suorittamisen apuvälineitä. Ohjelmoinnin opetuksen visualisointivälineet tarjoavat hyvän avun ohjelmoinnin opetuksessa käytettävien esimerkkiohjelmien sekä opiskelijoiden omien ohjelmien havainnollistamiseen. Opettaja voi hyödyntää näitä apuvälineitä omassa opetustyössään havainnollistamaan opetusta ja korvaamaan perinteistä ”kalvo-tussi-piirtoheitin”-kombinaatiota. Visualisointivälineet toimivat myös opiskelijan itsenäisen opiskelun tukena. Ohjelmointiharjoitusten suorittamisen apuvälineet puolestaan tarjoavat oppilaille valmiin ympäristön harjoitusten suorittamiseen. Tämä on tärkeää varsinkin ohjelmoinnin opiskelun alussa, sillä ohjelmointiympäristöjen rakentaminen harjoitusten tekemistä varten on usein hankalaa. Apuvälineissä ympäristöt löytyvät valmiina ja näin opiskelijat voivat keskittyä opiskelussa heti itse asiaan eli ohjelmointiin. Ohjelmoinnin apuvälineiden käytöstä hyötyvät sekä opettaja että oppilas.

Edellisen teoreettisen taustan jälkeen kuvattiin arviointimalli ohjelmoinnin apuvälineiden arviointiin. Erityisesti keskityttiin kriteereihin, jotka edesauttavat ohjelmoinnin perusteiden opetusta. Opettaja voi käyttää määriteltyä arviointimallia apuna etsiessään ja valitessaan sopivaa välinettä halumaansa käyttötarkoitukseen. Arviointimalliin määriteltiin muuttujia, jotka kohdistuivat välineiden soveltuvuuteen tiettyihin ennalta asetettuihin tarpeisiin. Tämän jälkeen käytiin läpi kaksi ilmaiseksi saatavaa esimerkkiapuvälinettä. Toinen, Jeliot 3, oli tarkoitettu visualisointiin ja toinen, Javala, oli tarkoitettu harjoitusten suorittamiseen.

Työssä pohdittiin myös sitä, miksi apuvälineitä käytetään hyödyksi niin vähän, vaikka niitä on saatavilla jopa ilmaiseksi. Syitä tähän olivat mm. välineistä puuttuva integrointi muihin opettajan käyttämiin välineisiin, niiden käyttöä ja käyttöönoton vaatimaa lisäresurssia ei ole huomioitu opetussuunnitelmassa ja pelko opetuksen muuttumisesta välinekeskeiseksi. Edellinen lista syistä kuitenkin osoitti, että syiden takana oli enemmän muut kuin tekniset seikat. Opettajat ovatkin avainasemassa näiden välineiden

käyttöönotossa. Tämän työn yhtenä tavoitteena olikin selventää välineiden taustoja ja mihin niitä voidaan käyttää.

Kehittämishankkeen tuloksista on toivottavasti hyötyä kaikille ohjelmoinnin opettajille ohjelmoinnin apuvälineiden valinnassa ja arvioinnissa. On huomioitava, että tässä työssä rajauduttiin vain niihin apuvälineisiin, jotka soveltuivat ohjelmoinnin perusteiden opetuksen ja opiskelun vaatimuksiin. Osa läpikäytyjen esimerkkivälineiden ominaisuuksista tuki pitemmällekin meneviä ohjelmointiominaisuuksia, mutta niitä ei tarkasteltu tässä työssä. Näiden läpikäynti olisikin hyvä tämä työn jatkoselvittelyn kohde. Myös laajempi esimerkkivälineiden ja varsinkin maksullisten apuvälineiden arviointi olisi arvokasta lisäselvitettävää.

Lähteet

Ala-Mutka, Kirsti 2006. Ohjelmoinnin opetuksen ongelmia ja ratkaisuja. Espoo: Teknillisen korkeakoulun Opetuksen ja opiskelun tuen julkaisuja 2/2006.

Boberg, Jorma, Lammi, Seppo, Penttonen, Martti, Salakoski, Tapio, Strömberg, Tuula & Teuhola, Jukka 2000. Johdatus tietojenkäsittelytieteeseen. Turku: Turun yliopisto, täydennyskoulutuskeskus.

Javala 2010. Javala – Java-oppimisympäristö. Tampereen teknillinen yliopisto. [online] [viitattu 1.12.2010] http://javala.cs.tut.fi/welcomePage.do

Jeliot 3 2008. Jeliot 3 kotisivu. Joensuun yliopisto. [online] [viitattu 1.12.2010] http://cs.joensuu.fi/jeliot/

Kaila, Erkki, Rajala, Teemu, Laakso, Mikko-Jussi & Salakoski, Tapio 2009. Tuloksia ja kokemuksia ohjelmasuorituksen visualisoinnista. Tietojenkäsittelytiede 29, 17–36.

Korhonen, Ari 2005. Visuaalinen algoritmisimulaatio ja sen sovelluksia. Tietojenkäsittelytiede 23, 42–59.

Levy, Ronit, Ben-Ari Mordechai & Uronen, Pekka 2003. The Jeliot 2000 program animation system. Computers & Education, 40, 1, 1–15.

Levy, Ronit & Ben-Ari, Mordechai 2007. We Work So Hard and They Don't Use It: Acceptance of Software Tools by Teachers. New York: In ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education, 246–250.

Malmi, Lauri, Karavirta, Ville, Korhonen, Ari, Nikander, Jussi, Seppälä, Otto & Silvasti, Panu 2004. Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2. Vilna: Informatics in Education, 3/2, 267–288.

McCracken, Michael, Almstrum, Vicki, Diaz, Danny, Guzdial, Mark, Hagan, Dianne, Kolikant, Yifat, Laxer, Cary, Thomas, Lynda, Utting, Ian & Wilusz, Tadeusz 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In ITiCSE-Working Group Reports (WGR), 125–180.

Truong, Nghi, Bancroft, Peter & Roie, Paul 2005. Learning to Program Through the Web. The 10th Annual Conference on Innovation and Technology in Computer Science Education Universidade Nova de Lisboa, ACM Press, 27–29.

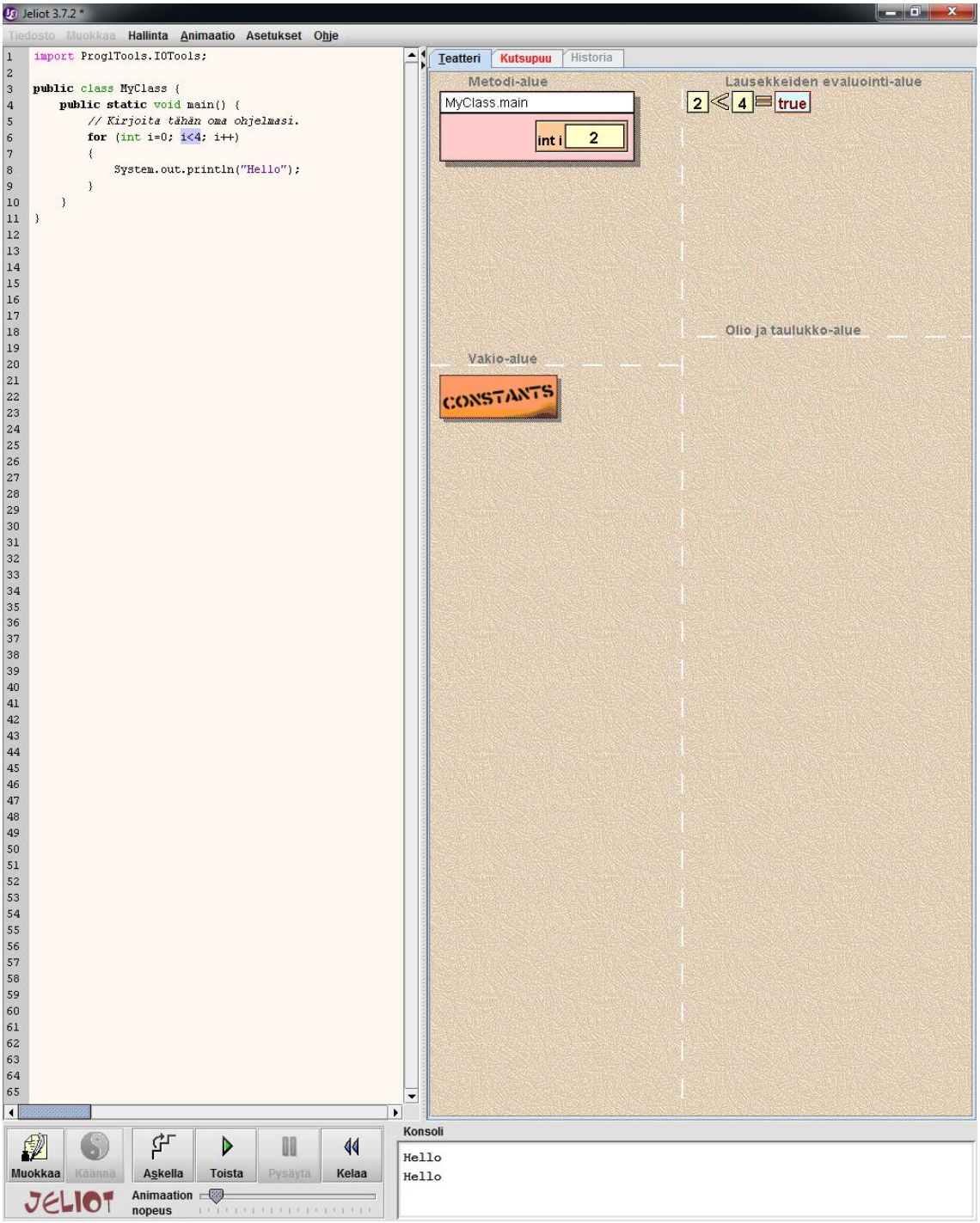
Wiggins, Melissa 1998. An overview of program visualization tools and systems. New York: 36th ACM annual Southeast regional conference, ACM Press, 194–200.

Wikla, Arto 2003. Ohjelmoinnin perusteet Java-kielellä. Hämeenlinna: OtaDATA.

Liitteet

Liite 1: Jeliot 3 ohjelman ikkuna (Jeliot 3 2008)

Ohjelmassa näkyy vasemmalla editori. Oikealla näkyy ohjelman visualisoimat metodikutsut, muuttujien arvot ja muutokset sekä operaatiot erillisessä omassa ikkunassa. Alhaalla oikealla on tulosteikkuna (konsoli), johon ohjelman tulosteet tulostuvat. (Jeliot 3 2008.)



Liite 2: Jeliot 3 käännösvirheiden esittäminen (Jeliot 3 2008)

Oikeassa ikkunassa oleva virheilmoitus perustuu Java-kääntäjän omaan virheilmoitukseen (Jeliot 3 2008).